Applying Large Language Models to Enhance the Assessment of Parallel Functional Programming Assignments

Skyler Grandel
skyler.h.grandel@vanderbilt.edu
Vanderbilt University
Nashville, Tennessee, USA

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
Vanderbilt University
Nashville, Tennessee, USA

Kevin Leach
kevin.leach@vanderbilt.edu
Vanderbilt University
Nashville, Tennessee, USA

ABSTRACT

Courses in computer science (CS) often assess student programming assignments manually, with the intent of providing in-depth feedback to each student regarding correctness, style, efficiency, and other quality attributes. As class sizes increase, however, it is hard to provide detailed feedback consistently, especially when multiple assessors are required to handle a larger number of assignment submissions. Large language models (LLMs), such as ChatGPT, offer a promising alternative to help automate this process in a consistent, scalable, and minimally-biased manner.

This paper explores ChatGPT-4's scalablility and accuracy in assessing programming assignments based on predefined rubrics in the context of a case study we conducted in an upper-level undergraduate and graduate CS course at Vanderbilt University. In this case study, we employed a method that compared assessments generated by ChatGPT-4 against human graders to measure the accuracy, precision, and recall associated with identifying programming mistakes. Our results show that when ChatGPT-4 is used properly (e.g., with appropriate prompt engineering and feature selection) it can improve objectivity and grading efficiency, thereby acting as a complementary tool to human graders for advanced computer science graduate and undergraduate students.

CCS CONCEPTS

• **Software and its engineering** → *Software maintenance tools*; • **Applied computing** → **Computer-assisted instruction**.

KEYWORDS

ChatGPT, Education, Generative AI, Large Language Models, Prompt Engineering, Automated Grading

ACM Reference Format:
Skyler Grandel, Douglas C. Schmidt, and Kevin Leach. 2024. Applying Large Language Models to Enhance the Assessment of Parallel Functional Programming Assignments. In *2024 International Workshop on Large Language Models for Code (LLM4Code '24), April 20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3643795.3648375

LLM4Code '24, April 20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0579-3/24/04…$15.00
https://doi.org/10.1145/3643795.3648375

1 INTRODUCTION

Conversational large language models (LLMs), such as ChatGPT-4 [13], have proven effective in a range of domains, including code generation and analysis [25]. LLMs are particularly promising in domains where humans and AI tools collaborate to more rapidly and reliably solve software problems [5, 24]. These advances have enabled the application of LLMs in educational domains, particularly in disciplines that benefit from automated and/or assisted analysis of textual or programming content [14, 19, 20].

Motivating the need for more effective and scalable CS program assessment tools. In the context of computer science (CS) education, assessing programming assignments is a task that traditionally requires a considerable expenditure of time and effort from instructors and/or graders. Moreover, the quality of the grading process is often susceptible to human error and subjectivity [15], particularly as class size grows. To mitigate such errors and accelerate the grading process, this paper explores the application of ChatGPT-4 to perform more objective and efficient analysis and assessment of student programs. This approach is increasing relevant as enrollments in CS classes increase, which often necessitates the use of multiple graders whose inconsistencies (known as the "inter-rater reliability problem") can pose substantial challenges for fair and reliable grading [7, 15].

To cope with the impact of scale in large CS classes, programming assignments are often assessed via automated graders, which are similar to unit and/or integration test suites. While automated graders are useful in helping to assess functional correctness, they offer limited aid in judging programming style, efficiency, and other quality attributes [4, 6, 15]. Moreover, automated test suites may stifle student creativity by mandating overly restrictive structures to fit within the "Procrustean Bed" of auto-graders.

In the case of coding style, many instructors adopt "linters" to identify stylistic mistakes [10]. However, these tools are limited in their ability to capture certain elements of coding style, such as documentation completeness or holistic readability. In contrast, LLMs, such as ChatGPT-4 or Claude, offer a more flexible qualitative grading solution that can evaluate functionality, coding style, efficiency, and other quality attributes in a largely automated fashion.

More generally, the integration of generative AI tools into CS pedagogical practices can pave the way for more personalized and adaptive learning experiences through bespoke feedback that conventional unit and integration tests cannot provide [2, 4, 6, 15]. This paper thus presents the results of a case study conducted in an upper-level undergraduate and graduate course at Vanderbilt University entitled "Parallel Functional Programming in Java".[1] This

[1]All course material is available at `www.dre.vanderbilt.edu/~schmidt/cs253` in open-source form.

case study explores how the application of LLMs in grading and programming assignment assessment through a semi-automated grading methodology using ChatGPT-4 can supplement and enhance conventional automated graders and traditional manual grading.

Solution approach → The GreAIter LLM-based auto-grading tool. We rely on prompt engineering techniques [24] to converse effectively with ChatGPT-4. A prompt is a set of instructions provided to an LLM that programs the LLM by customizing it and/or enhancing or refining its capabilities, which can influence behavior of and interactions with LLMs [25]. Prompt engineering is the means by which LLMs are programmed via prompts, guided by experience [23, 26] on applying LLMs effectively. We applied these techniques to develop an LLM-based auto-grading tool (known as "GreAIter") that assists human graders in locating faults and generating accurate and meaningful feedback for students.

Our case study implemented GreAIter using ChatGPT-4 and applied it to evaluate programming assignments in the "Parallel Functional Programming in Java" course. We defined rubrics using a structured JSON format with specific grading criteria to communicate the grading methodology to ChatGPT-4. For each criterion contained in the rubric, ChatGPT-4 was instructed to

(1) Output the code from each student submission that is relevant to that criterion and a grade of "correct" or "incorrect" and then
(2) Compile a summary of all mistakes made by the student and outputs suggested feedback for the student based on their submission and the mistakes therein.

The results of this assessment was reviewed by human graders to produce the student's final grade, thereby yielding an efficient, accurate, and minimally-biased grade due to the collaboration between humans and the GreAIter generative AI tool.

Evaluation approach and research contributions. To evaluate our approach, we assessed ChatGPT-4's performance in grading programming assignments and compared this performance to that of human graders. We conducted this comparison by examining ChatGPT's accuracy and efficiency in evaluating student submissions given a rubric compared to a human grader. We investigated the false positive and false negative rate through precision and recall to determine ChatGPT-4's shortcomings as a grader and analyzed how it can be used in a semi-automated approach. We further examined the objectivity of results to assess ChatGPT-4's potential in reducing subjective bias in grading by comparing results from multiple grading attempts given the same submission.

This paper provides the following contributions to research on AI-assisted programming assignment evaluation:

- It provides empirical evidence regarding the utility of ChatGPT-4 as a tool for assistance in assessing programming assignments for advanced computer science topics, *i.e.*, parallel functional programming.
- It offers insights into the extent to which LLMs like ChatGPT-4 can be relied upon for accurate assessment in educational settings, potentially setting the stage for broader adoption and further technological development.

Paper organization. The remainder of this paper is organized as follows: Section 2 describes the methodology of our work, encompassing the design of our GreAIter auto-grading tool and the prompting strategies we used to achieve our results; Section 3 explains the experiment we designed to assess the performance of our methodology in grading programming assignments in the "Parallel Functional Programming in Java" course; Section 4 evaluates the results of our GreAIter grading tool using ChatGPT-4; Section 5 explores the limitations and threats to validity of our work, Section 6 compares our research with related work on AI-assisted programming assignment evaluation; and Section 7 presents concluding remarks and future work.

2 METHODOLOGY

This section describes the methodology of our study, focusing on the design of our LLM-based GreAIter auto-grading tool and the prompt engineering strategies we applied to achieve our results.

2.1 Overview of GreAIter and our AI-assisted Grading Process

Figure 1 depicts the steps involved in our grading process using GreAIter. We begin with the student submission (1) and the rubric

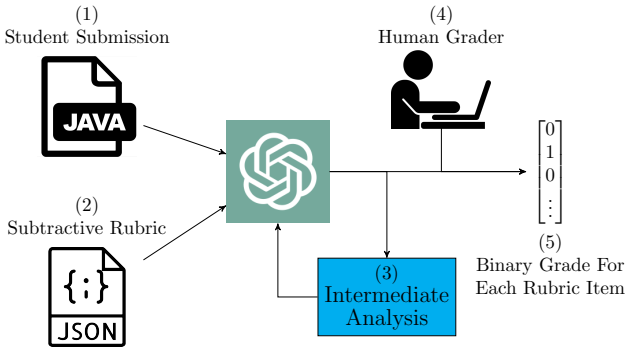

Figure 1: GreAIter's AI-Assisted Grading Process.

being input (2) into ChatGPT-4. This LLM then conducts an intermediate analysis (3), consisting of a detailed evaluation for each criterion in the rubric. ChatGPT-4 then summarizes its assessment and a human grader reviews the output (4), verifying and adjusting ChatGPT-4's evaluations as needed. The final output from the human grader (5) consists of a binary grade for each criterion in the rubric, indicating the performance of the student's programming assignment submission against each criterion.

GreAIter provides a bridge between theoretical generative AI capabilities and practical educational applications. This tool enables instructors to use LLMs effectively to improve grader efficiency and objectivity by providing an automated system that interprets student submissions against a predefined rubric and produces an objective assessment mirroring human graders' processes. GreAIter leverages *rubric-based evaluation*, where each criterion is clearly defined via a structured format. This format provides ChatGPT-4 with the parameters needed to assess student submissions and ensure consistency across multiple evaluations, thereby addressing the inter-rater reliability problem [7].

While GreAIter contains no elements that are specific to any particular programming assignment, we recommend certain steps

for integrating it into a class.[2] In particular, a reliable and structured rubric is necessary for effective results. ChatGPT-4 has been shown by others [27] as an adequate prompt engineer on the level of humans, and we leverage this capability to generate rubrics that GreAIter uses to prompt ChatGPT-4. In particular, ChatGPT-4 can generate a usable rubric given a (1) list of potential mistakes, (2) an "answer key" (*i.e.*, the desired assignment solution), and (3) the JSON structure shown in Figure 2. We follow this approach to generate the rubrics for our experiments, with researchers verifying each rubric criterion that ChatGPT-4 outputs to ensure quality of description and correct and incorrect examples.

Our initial round of experiments indicated that ChatGPT-4 was prone to generating code for the incorrect example in a rubric criterion that is different than real mistakes students may make. For example, we might want ChatGPT-4 to ensure that students use Java method references rather than lambda expressions where possible for stylistic reasons. Here, the desired solution might look something like, ".map(this::someFunction)", while ChatGPT-4 might generate something like ".map(item ->)" instead of "map(item -> someFunction(item))", which is semantically equivalent to the correct answer. We therefore provided examples of incorrect code in prompts to generate these rubrics and verified the incorrect example to ensure it exhibited realistic mistaken behavior.

2.2 Prompt Engineering and Human-AI Collaboration

While GreAIter is capable of operating in a fully autonomous mode, we applied a semi-autonomous method due to limitations associated with current LLM technologies, including ChatGPT-4 [3, 23–25]. Despite its advanced capabilities, ChatGPT-4 can generate errors (commonly referred to as "hallucinations"), where it confidently asserts inaccurate or nonsensical information [3]. This tendency is problematic for educational assessments, where the stakes of incorrect evaluations are high, as they may significantly impact a student's learning trajectory and academic record.

Given a programming assignment and rubric, GreAIter generated feedback for human graders to review. As a final sanity check, human graders then checked the relevant segment(s) of student code identified by GreAIter to manually verifying the issues it flagged were indeed mistakes (rather than false positives). Human graders thus scored each student appropriately and reviewed GreAIter's feedback before returning results to students via GRADE files pushed to their GitLab repositories.

By integrating a human-in-the-loop approach, we introduced a crucial verification step. Human graders review the AI-generated assessments, ensuring the reliability of the final output. This safeguard is not merely a corrective measure, it also reinforces the educational value of the grading process. A human grader's oversight ensures that feedback is pedagogically appropriate and contextually relevant to each student's learning needs.

Our semi-automated approach also aligns with ethical guidelines, promoting responsible AI by mitigating risks associated with unverified autonomous AI operation in high-stakes application domains like primary and secondary education. Our approach respects the sophistication of the AI while prudently managing its limitations and balancing the efficiency of automation without foregoing the expertise of human educators. The result is a hybrid model that aims for high-quality, scalable assessment mechanisms that both educators and students can reply upon.

At the heart of GreAIter's functionality is prompt engineering, *i.e.*. the intentional design of prompts that guide LLMs in performing their tasks. For our GreAIter process, prompts are carefully crafted to elicit specific behaviors from ChatGPT-4, enabling it to understand and apply the grading rubrics accurately. Due to ChatGPT-4's familiarity with JSON, we found that formatting the rubric using JSON (as shown in Figure 2) is an effective prompting strategy to

```
{"[Filename]": {
    "criteria": [
      {
        "title": "[Title]",
        "description": "[In depth description
            of the criterion]",

        "correct_example":
        "[Correct example in your assignment's
            programming language]",

        "incorrect_example":
        "[Incorrect example in your assignment's
            programming language]"
      },
      .
      .
      .
    ]
}}
```

Figure 2: Example JSON to Provide a Rubric to ChatGPT-4.

ensure ChatGPT-4 accurately parses the information in the rubric. We therefore define our rubric as a JSON array where each element contains an object representing a rubric criterion. Each rubric criterion contains entries for the criterion's title, description, and a correct and incorrect example.

We used the following prompt to instruct ChatGPT-4 on the use of this rubric:

> You are a grader for the parallel functional programming course taught in Java. I will give you a JSON rubric and student Java code. For each item in the rubric, you will first output the function in the student's code that is relevant to that item and then you will output a score of "correct" or "incorrect". Alternative answers to the correct code are permissible if they have the same functionality and do not apply poor style conventions.

This prompt was followed by the rubric and the student's code. A subsequent request instructed ChatGPT-4 to compile a comprehensive summary of errors or misalignments with the rubric's expectations, along with suggested feedback for the student based on their specific mistakes. This prompt forced ChatGPT-4 to consider each individual criterion in the rubric, and then used a chain-of-thought[3] prompting strategy by asking the LLM to output the relevant code before making a judgement about its correctness. These strategies

[2] GreAIter is available to instructors of CS programming courses on request.

[3] Chain-of-thought prompting [23] instructs an LLM to explain its "thought process" before giving an answer to improve answer quality.

helped to minimize ChatGPT's tendency to hallucinate errors, skip rubric criteria, and/or consider irrelevant parts of student code.

Based on our experience with the GreAIter case study, including both correct and incorrect examples in the rubric is crucial for several reasons. First, it enables a "few-shot learning"[4] method that provides context and clarifies edge cases and potentially ambiguous instructions. Second, these examples aid ChatGPT-4 in providing specific feedback to students by comparing their solutions to the desired solution.

2.3 Assessment Process and Ethical Considerations

Our GreAIter-based assessment process began with ChatGPT-4 receiving each student's code and the associated rubric through our prompts, as shown by step (1) in Figure 1. ChatGPT-4 then systematically evaluated the code, criterion-by-criterion, referencing specific code segments as evidence for its assessments. Our prompts were designed to ensure that ChatGPT-4's evaluation was not merely keyword-based but contextually rooted in the logic and syntax that the rubric required.

Upon completion of the assessment for each criterion, ChatGPT-4 aggregated individual assessments into a final summary. This summary conveyed areas where the student excelled, as well as areas that require further improvement. In addition, this summary provided a foundational tool for human graders to either validate the results of GreAIter's grading process or to provide additional insights where necessary.

To maintain the integrity of the assessment, we also included a review mechanism where the outputs generated by GreAIter were cross-examined by human graders. This dual-layered approach not only fine-tuned the assessment process but also established a comprehensive feedback system that benefited the students' learning experience. Our goal was to harness the computational precision and scalability of ChatGPT-4 while retaining the nuanced judgment of humans, striving for an equilibrium that augments the grading process within Vanderbilt University and other educational environments.

3 EXPERIMENT DESIGN AND EVALUATION

To evaluate our methodology described in Section 2, we designed an experiment to empirically determine how well ChatGPT-4 performed in its assessment of student parallel functional programming assignments. The objective of this experiment was to assess the performance of a ChatGPT-4-based automated code assessor against human graders in terms of accuracy, efficiency, and objectivity. The experimental setup described in this section measured the efficacy of GreAIter by comparing its assessment outcomes to those of experienced human graders.

Assignments and student submissions for this experiment were obtained from our parallel functional programming course at Vanderbilt University, which consisted of 26 undergraduate and graduate students in the fall semester of 2023. We considered results from three assignments given to this class cohort and used final student grades on the assignments as the "ground-truth" human-graded benchmark. To eliminate inter-rater bias, one graduate teaching assistant (TA) grader with five years of experience with Java and two years of experience with Java parallel functional programming initially graded all submissions. These grades were then reviewed by the course instructor to ensure accuracy.

The TA grader was given the same rubrics as GreAIter, though these rubrics were formatted as plain text instead of JSON for readability. We recognized that a single TA grader reviewed by a single instructor might exhibit potential bias, so we investigated each inaccuracy carefully to determine the cause of potential flaws.[5]

Our experiment considered the following three research questions:

RQ1: Performance Can GreAIter perform correctly by identifying mistakes in student program submissions?

RQ2: Efficiency What is the reduction in the amount of manual grading that must be done when using GreAIter compared to traditional manual grading?

RQ3: Consistency How consistent is our LLM grading methodology across multiple grading attempts of the same programming assignments?

Section 4 below discusses our recommendations for integrating our GreAIter grading methodology as a semi-automated grader in CS classes. For this experiment, however, we evaluated GreAIter's performance in isolation to provide evidence for our recommendation. While GreAIter could be used to fully automate grading, we use our experiments to determine GreAIter's failure modes to evaluate its efficacy and determine how an AI-assisted grader can verify results.

3.1 RQ1: Performance

Building upon the experimental design described above, we used three performance metrics to evaluate GreAIter rigorously. First, we used GreAIter's *accuracy*, which we quantified as the percentage of student mistakes correctly identified by GreAIter in alignment with the consensus grades established by the TA grader and instructor. High accuracy results would validate GreAIter as a reliable evaluator of code quality and correctness, thereby motivating its integration into the grading process to reduce the grading load on instructors while maintaining high assessment standards.

Second, we used GreAIter's *precision*, which we quantified as ChatGPT-4's tendency to incorrectly mark a correct code segment as erroneous. This metric is crucial because high precision indicates GreAIter rarely marks correct code as erroneous, preventing undue penalties on students and reducing the need for human oversight. High precision indicates GreAIter's meticulousness, ensuring its feedback is constructive and based on actual student errors, thereby maintaining student trust in this assessment process. While poor precision would impact ChatGPT-4's ability to operate in isolation, it could be mitigated with additional TA grader intervention as part of the overall GreAIter assessment process.

Third, we use *recall*, which we qualitifed as GreAIter's ability to identify all incorrect code present. A high recall rate indicates GreAIter can effectively detect most—if not all—errors in student

[4]Few-shot learning involves training an AI model from only a few examples, which typically allows the model to perform better than it would in a 0-shot approach [22], which has proven effective for LLMs [12].

[5]We assumed that a three-way agreement between the TA grader, the instructor, and GreAIter is most likely accurate, so we do not investigate cases where such a consensus is reached.

submissions, which is critical because identifying candidate mistakes demonstrates ChatGPT-4's ability to assist human graders. In contrast, a low recall rate would require human oversight to a degree that GreAIter would not substantially accelerate the assessment process, particularly at scale as CS class sizes increase. If GreAIter exhibited high recall its utility in providing comprehensive and thorough feedback necessary for educational purposes is enhanced, *i.e.*, it can consistently identify mistakes (which can then be verified quickly by human graders).

Table 1: Confusion Matrix of LLM-Produced Grades.

AI Grader Results Compared to Human Graders		
	True	False
+	**2.61%**	**1.79%**
–	**95.60%**	**0%**

A summary of intermediate statistics is shown in Table 1, which depicts positives (+) and negatives (–) found by GreAIter and shows promising statistics with a potential failure mode being present in false positives, to which GreAIter is prone. The accuracy of GreAIter was benchmarked against the grades determined by the TA grader and validated by the course instructor, which yielded the following results:

- **True Positives (TP): 2.61%** - The percentage of instances where GreAIter correctly identified errors that were also recognized by the TA grader.
- **True Negatives (TN): 95.60%** - The percentage of instances where GreAIter correctly identified correct code segments, aligning with the TA grader's assessments.
- **Overall Accuracy: 98.21%** - The proportion of correct assessments made out of all grading decisions.

The precision and recall of GreAIter reflected its ability to minimize false positives and false negatives, both crucial aspects of ensuring fair and constructive feedback:

- **False Positives: 1.79%** - The percentage of instances where GreAIter marked correct code segments as erroneous.
- **False Negatives: 0%** - The percentage of instances where GreAIter marked erroneous code segments as correct.
- **Precision: 59.30%** - GreAIter's ability to correctly identify student mistakes without over-penalizing correct aspects of their submissions.
- **Recall: 100%** - The comprehensiveness of GreAIter in detecting errors present in the student submissions.

The results of applying our semi-automated GreAIter tool revealed a nuanced performance profile across the metrics presented above. Our high overall accuracy rate indicates that GreAIter aligned well with the TA grader performance in the vast majority of cases. This finding suggests a strong foundational reliability of ChatGPT-4 in evaluating parallel functional programming assignments.

Interestingly, the seemingly low true positive rate of 2.61% indicates we had somewhat skewed data, with considerably more true negatives than true positives. This result is not unexpected since student grades on programming assignments in this course tend to average over 90%. However, it does render our true positive rate somewhat meaningless.

A precision of 59.30% implies that when GreAIter identifies an error, it is correct just over half of the time, indicating a tendency towards false positives, where GreAIter incorrectly marks code as erroneous. While GreAIter is thorough, therefore, it may be overly critical or prone to hallucination, *i.e.*, by perceiving errors that are not there. More optimistically, GreAIter exhibited perfect recall in this trial, meaning it never missed a mistake made by a student.

Overall, these results suggest that while GreAIter shows promise in terms of high accuracy and recall, its should be managed carefully due to its propensity for false positives. The strong recall indicates that GreAIter can serve as an effective initial filter in identifying potential errors in student submissions. However, the precision underscores the necessity of human oversight to confirm ChatGPT's findings and to provide the final judgment on the student's work, which substantiates our focus on a semi-automated grading approach.

3.2 RQ2: Efficiency

To address the second research question, we focused on evaluating the efficiency of our LLM-based grading system compared to traditional manual grading methods. In this context, we defined efficiency by (1) the time investment required for grading, (2) the number of rubric criteria a grader must assess, and (3) the volume of code that must be reviewed for each submission.

3.2.1 Time Investment. Based on our experience applying GreAIter throughout the fall semester of 2023, we found that the semi-automated process for grading was notably faster than manual grading. In particular, we observed that our grader's runtime average roughly one minute per student submission due to the request latency of the OpenAI API. However, GreAIter could simultaneously assess *all* submissions for a given assignment, so it could run as a background process while the TA grader reviewed the results. The runtime of GreAIter thus had a negligible effect on overall grading efficiency.

The time needed to grade each submission is a critical measure of efficiency. We tracked the duration it took for a human grader and our semi-automated GreAIter approach to complete the grading process for all student submissions in a given assignment. Our observations indicated that GreAIter substantially reduced the time required per submission. In particular, the average time taken by GreAIter to assess all submissions for a single assignment was roughly 45 minutes, or 1.73 minutes per student submission.

In contrast, the TA grader spent an average of just under 4 hours to grade all submissions for an assignment, or 9.23 minutes per submission. We therefore found that our approach reduced overall grading time by approximately 81.2%, which highlights the potential of GreAIter to enhance grading efficiency in educational settings. This increased efficiency would be even more apparent in much larger CS classes that required multiple TA graders, and would also address the inter-rater reliability problems that would likely arise with multiple TA graders.

3.2.2 Number of Rubric Criteria. Another dimension of efficiency we investigated was the number of rubric criteria that a grader must check. In the traditional manual method, a TA grader must assess each criterion for every submission. In contrast, GreAIter's ability to quickly identify correct code segments reduced the number of

criteria requiring detailed review. We quantified the average number of rubric criteria the TA grader had to assess in-depth for each submission compared to those GreAIter those flagged for further review. Our findings showed that GreAIter required attention to roughly 1.1 rubric criteria per student submission on average, which was substantially less than the average of 25 criteria for the TA grader. Overall, this constituted a substantial decrease of 95.6%.

3.2.3 Volume of Code. Finally, we evaluated efficiency in terms of the total number of lines of code a grader needed to check to grade a submission. GreAIter's ability to precisely target relevant code segments for each rubric criterion reduced the overall volume of code that needs in-depth review. This property of GreAIter resulted from its ability to find the relevant method to each rubric criterion within student code. GreAIter thus only needed to check methods that ChatGPT-4 highlighted as relevant to a rubric criterion that a student missed.

We compared the average number of lines of code reviewed per submission by GreAIter and the TA grader. GreAIter reviewed an average of 28 lines of code per student submission. In contrast, the TA grader reviewed an average of 409 lines of code, constituting a 93.2% decrease in volume.

The results from our experiments indicate a substantial improvement in grading efficiency when using ChatGPT-4 as the LLM for the GreAIter process. The reduction in time per-submission—coupled with fewer rubric criteria requiring in-depth review and a lower volume of code to scrutinize—significantly reduced our manual grading workload. This efficiency does not come at the cost of grading quality since GreAIter still adhered to the grading standards we established. By freeing up time and resources, moreover, we could focus more on providing quality feedback, engaging in interactive teaching, and developing better course content, thereby enriching overall student educational experience.

3.3 RQ3: Consistency

To assess the reliability of GreAIter, we implemented a repeatability test, which provided a crucial measure of our grader's consistency over time. In this context, consistency refers to the ability of GreAIter to produce the same results when presented with the same inputs under similar conditions. This ability is vital for its potential deployment in educational settings and ensures our tool's accuracy results are not the result of a chance response from ChatGPT-4.

The primary metric for success in the repeatability test is the *consistency rate*, *i.e.*, the percentage of identified mistakes that remain unchanged across multiple grading attempts by GreAIter. A high consistency rate indicates that GreAIter is stable and reliable in both its grading and feedback and exhibits minimal inter-rater reliability bias, which is essential for any (semi-)automated grading tool used in academia.

The consistency rate we observed was **78%**, which was the ratio of rubric criteria for which GreAIter gave the same result to the same student submission over two grading attempts. However, all identified disagreements were false positives, *i.e.*, in one trial GreAIter hallucinated a problem while it did not hallucinate or hallucinated a different problem in the other trial. This finding highlights the stability of GreAIter and the minimization of grader bias.

This consistency rate informs us about the repeatability of GreAIter's performance. Although this rate is not perfect, it is substantial and indicates that GreAIter can reliably reproduce its grading decisions across multiple iterations. The inconsistencies we observed were due to false positives, reinforcing our earlier observation regarding GreAIter's tendency to over-diagnosis errors. The nature of GreAIter's inconsistencies are thus consistent with our previous findings that underscore the necessity of human oversight to confirm GreAIter's findings and to provide the final judgment on the student programming submissions.

This semi-autonomous "augmented intelligence" approach (*i.e.*, where GreAIter provides a first pass and humans verify) offers a balanced solution, combining the thoroughness and speed of LLMs like ChatGPT-4 with the discernment and expertise of human graders. This hybrid strategy helped us streamline the grading process, reduce the workload for instructors and TAs, and maintain the integrity and fairness expected of academic evaluations.

4 ANALYSIS OF RESULTS

This section analyzes the results of our semi-automated GreAIter tool, focusing on the implications of its performance metrics, its potential role and integration within educational settings, and considerations for its future application. A summary of our experimental results using the TA grader as the ground truth is presented in Table 2.

Table 2: Summary of Results.

Performance	Accuracy 98.21%	Precision 59.30%	Recall 100%
Effort Reduction	Time 81.2%	Rubric Criteria 95.6%	Code Volume 93.2%
Consistency		Consistency Rate 78%	

Performance metrics and efficiency gains. GreAIter's high overall accuracy (98.21%) and recall (100%) indicate its potential as a useful tool and process in programming assignment assessment and grading. Its effectiveness in correctly identifying correct code submissions significantly reduced TA grader workload by filtering out submissions that likely required no further review. However, the precision of 59.30% raises concerns regarding its number of false positives. This result reflects the current limitations of ChatGPT-4, which, while sophisticated, can still misinterpret complex instructions or code nuances, leading to the incorrect identification of errors.

This outcome led us to applying a semi-automated grading and feedback approach, which leverages GreAIter to find candidate mistakes in student code for subsequent TA grader review. Since GreAIter outputs the relevant code for each mistake, TA graders can quickly validate candidates. This semi-automated process can accelerate the grading process and reduce TA grader effort, as shown by our efficiency study in Section 3.2, while maintaining human intervention to mitigate student distrust of AI-based systems.

Repeatability and semi-automation. The repeatability rate of 78% suggests that while GreAIter is generally reliable, there are some variations in its grading across iterations. This variability

is particularly problematic in the context of CS courses, where consistency in grading programming assignments is paramount to fairness and credibility of the assessment process. However, because the disagreement between trials stems entirely from false positives, we can use GreAIter to focus TA grader attention on reducing bias and improving grading efficiency by decreasing grading time by up to 81.2%.

These results also indicate that GreAIter can contribute effectively to the grading process, primarily through initial assessments and identification of clear-cut cases of correct code. Its high accuracy in these instances could enable instructors to allocate more time to providing in-depth feedback where it is most needed, potentially enhancing the educational experience for students. Nevertheless, GreAIter's propensity for false positives necessitates a semi-automated approach where human graders perform a secondary review of its grading decisions. This approach leverages the strengths of GreAIter in rapidly processing and evaluating submissions while mitigating its weaknesses through human oversight.

5 LIMITATIONS AND THREATS TO VALIDITY

This section explores the limitations of the GreAIter AI-assisted grader described in Section 2 and the threats to validity of our experiments described in Section 3.

Generalizability across languages and paradigms. Although our study is extensive, it is not without limitations. In particular, we developed and evaluated GreAIter within a single course (parallel functional programming) and programming language (Java), which may limit the generalizability of our findings. For example, Java programming, and more specifically, the parallel functional programming paradigm, has some unique challenges and patterns that may not be representative of other programming languages or paradigms.

Subjectivity in ground truth and precision concerns. We compared GreAIter's performance to that of a single TA grader and course instructor, which may introduce a degree of subjectivity into the "ground truth" against which GreAIter's performance was measured. Although the TA grader was quite experienced—and all grades were reviewed for accuracy—different graders may have different thresholds for correctness and error severity, potentially influencing the benchmarks used for AI evaluation. It may therefore be prudent to verify our results with multiple graders and courses, along with a comparison to human inter-grader reliability.

The false positives reported in GreAIter's outcomes reflect another limitation. While its recall was perfect (indicating it missed no errors) its precision was much lower, suggesting GreAIter sometimes identified errors to zealously. While this over-detection erred on the side of caution, it led to unnecessary reviews by human graders, diminishing the efficiency gains from using GreAIter.

Consistency and repeatability concerns. Another potential threat to the validity of the study is our reliance on the OpenAI API's latency, which may affect the grading speed results. Likewise, the consistency measure assumes that ChatGPT-based LLMs will not learn or adapt over time, which may not hold true as its underlying models are continuously updated and improved upon. However, the importance of this metric is in verifying the consistency of grades within a single cohort for a single assignment, and updates to the base model should not be variable within a single assignment. Furthermore, the repeatability test, while designed to be rigorous, was confined to two trials. More extensive testing over additional trials could provide a deeper understanding of GreAIter's consistency and reliability.

Despite these limitations and threats to validity, our study provides valuable insights into the capabilities and limitations of using ChatGPT-4 for assessing programming assignments. The high accuracy and recall offer evidence of GreAIter's potential utility in CS courses, and its consistency rate, is promising, though imperfect. The careful design of our study, the systematic approach to data collection and analysis, and the critical evaluation of results all contribute to the robustness of our findings. These limitations provide a clear framework for understanding the context within which the findings are applicable.

6 RELATED WORK

LLMs and AI-assisted education are a active areas of research that we build upon by applying prompt engineering techniques to LLMs to facilitate automated assessments of student programming assignments. This section compares our research with related work in the fields of prompt engineering and AI-assisted education.

Several studies have made use of prompt engineering to improve the performance of LLMs, from simple prompt strategies [1] to more complex ones [24, 26]. Wei et al. [23] have investigated "chain-of-thought" prompting, which is an approach we apply in our work. Yao et al. have improved on this work by including action plan generation and external source lookup [26]. White et al.[24] developed prompt patterns, analogous with software patterns, that can improve and structure LLM outputs and they have followed this with a study on prompt patterns for improving code quality in particular [25]. Common failure modes for LLMs have been identified as well, necessitating the development of improvements and mitigations [3].

Suggestions for future use of LLMs guide our study as well. We strive to follow guidelines put forth by van Dis et al. [21], particularly suggestions to "embrace the benefits of AI" and "hold on to human verification". We leverage insights from this and other related work to develop a novel programming assignment assessment methodology to speed up grading and mitigate inter-grader bias.

Other researchers have explored applications of AI in education specifically. Most similar to our work is a study on automated grading of short answer questions using LLMs [19], which also found that an AI grader necessitates human oversight, though it is still helpful for maximizing grading efficiency. LLMs were applied for short answer grading as a followup [18] to a study on the same task using fine-tuned transformer models, which outperformed other automated attempts but did not achieve sufficient accuracy for full integration. Several studies [2, 9, 16, 20] have speculated on the benefits and pitfalls of LLMs and AI in education to guide future research, such as our study.

Detection of AI-generated submissions for education [14], generation of programming exercises and code explanations [17], and assistance in medical education using LLMs [11] have also been studied. This related work showcases the potential of LLMs for education, which helped guide our study. Finally, there is a long tradition of automated grading of programming assignments using test suites and linting for correctness and style [6, 8, 15]. We build

on this related work by using ChatGPT-4 to overcome limitations in qualitative assessment, thereby enabling automated assessment of efficient implementations, adequate documentation, and a broader range of stylistic issues.

Our study builds upon the rich and growing body of research in the realms of prompt engineering, AI-assisted education, and automated grading systems. We acknowledge the various methods and applications explored in these fields, ranging from improving LLM outputs through intricate prompt designs to leveraging AI for educational purposes. Our approach contributes to this evolving landscape by applying sophisticated prompt engineering techniques to GPT-4 for the specific task of assessing programming assignments. This novel application not only addresses the challenges of efficient and minimally-biased grading but also encapsulates the potential of AI in enhancing the educational experience.

7 CONCLUDING REMARKS

This paper presented the results of our study that applied ChatGPT-4 to create GreAIter, which is an AI-assisted tool that helps automate key portions of the grading process for programming assignments in an advanced parallel functional programming course offered in Java. Our findings codify the potential and current limitations of AI-assisted grading systems and yielded the following lessons learned:

- **ChatGPT-4 has the capacity to accurately identify correct code submissions.** We demonstrate that GreAIter could achieve a high accuracy rate (98.21%) and perfect recall. These results suggest that LLMs can play an important role in assisting with grading tasks, particularly in filtering out rubric criteria that are likely correct for a given submission, thereby reducing human grader workload. Ironically, when ChatGPT-4 did not accurately identify correct code submissions, we interacted with it and got it to explain how we could craft future prompts to elicit more accurate results from it. The ability to engage in such a "Socratic dialogue" with an LLM like ChatGPT-4 was quite refreshing compared with traditional means of refining queries with conventional static analysis tools.
- **The need for human oversight remains critical.** The precision of 59.3%, marked by a substantial rate of false positives, points to the limitations of the current state of LLMs in understanding and evaluating complex programming tasks. Therefore, despite GreAIter's impressive recall rate (indicating no missed errors) human oversight remains necessary due to ChatGPT-4's tendency to over-flag student code segments as erroneous. By integrating insights from previous work with GreAIter, we extend the capabilities of LLMs in educational contexts and set the stage for future research in AI-assisted education. The synergy of LLMs and human expertise demonstrated in this study showcases the potential of LLMs in enhancing educational methodologies and outcomes.
- **Due to ChatGPT-4's limitations, we stress the benefits of a semi-automated grading approach.** A key lesson learned through our study is the importance of human-AI collaboration, which is commonly known as "augmented intelligence" rather than conventional "artificial intelligence". While GreAIter is powerful, it is not yet capable of replacing human judgment in tasks that require nuanced understanding. The semi-automated approach advocated by our research—where ChatGPT-4 performed an initial assessment and humans provide final verification and feedback—strikes a balance that leverages the strengths of both. We found this semi-automated approach reduced grading workloads, constituting a 93.2% decrease in code volume to review and an 81.2% decrease in grading time. We also found GreAIter yielded a consistency rate of 78%, thereby indicating that while LLMs can exhibit consistent, relatively unbiased tendencies, their performance can vary, and thus should be regularly checked for consistency and accuracy.
- **GreAIter can be integrated into actual classroom settings to improve grader efficiency and reliability.** Through careful planning and systematic analysis of the accuracy, precision, recall, efficiency, and repeatability metrics covered in this paper, we showed that GreAIter improves traditional TA grading. We validated the feasibility of integrating GreAIter into an actual classroom setting, optimizing the grading process in terms of both efficiency and scalability. While GreAIter demonstrates a high degree of accuracy, its current limitations underscore the need for a semi-automated approach that combines the speed and consistency of LLMs with the critical thinking and expertise of human graders. As LLMs evolve, so too will the strategies for integrating it more effectively to enhance quality and fairness of the grading process for CS courses.

Overall, our case study shows that the promise of LLMs in education extends beyond grading efficiency, *i.e.*, LLMs have the potential to reshape how feedback is delivered, how learning is assessed, and how education is ultimately conducted. While LLMs have not yet reached the point of replacing human graders, they provide an important resource to aid educators, particularly in disciplines like CS that are characterized by ever-growing class sizes. In keeping with previous research [24, 25], we find that collaboration between human users and AI tools results in rapid and reliable software solutions, and we leverage this collaboration for a more efficient and effective educational process. As LLMs grow more sophisticated, we anticipate further research to refine and harness these powerful tools for the betterment of educational systems.

Looking forward, the integration of LLMs into grading CS programming assignments requires consideration of the trade-offs between efficiency and accuracy. The false positive rate must be reduced to make tools like GreAIter more autonomous and trustworthy. Our future work will focus on fine-tuning LLMs on larger and more diverse datasets of code submissions and rubrics, potentially improving their understanding and reducing the rate of false positives. Likewise, we plan to explore the use of ensemble methods, which combine multiple LLMs and/or AI-assisted graders to cross-verify results and improve grading consistency. Many false positives result from the same rubric criteria, so investigations into prompting strategies for specific rubric criteria is also likely to improve precision.

The limitations described in Section 5 also offer directions for future research. For example, our future work will explore a wide range of courses, broader coverage of programming languages,

more diverse grading benchmarks, as well as other LLMs beyond ChatGPT-4. By understanding the specific contexts in which GreAIter performs well, and those in which it does not, we can better tailor the this type of AI-assisted grader to meet instructor needs. Thus, while our case study is bound by certain limitations and potential threats to validity, our methodology and the GreAIter's overall solid performance in several key metrics support the validity of our findings. The study's design and the presented results provide a foundation upon which future work can build, contributing to the evolving field of LLMs in CS education and the development of more sophisticated, reliable, and efficient AI-assisted graders.

8 ACKNOWLEDGEMENTS

We applied ChatGPT-4 extensively to implement GreAIter and perform the automated assessment of student programming assignments described in this paper. We also applied ChatGPT-4 to check the spelling and grammar of this paper.

REFERENCES

[1] Simran Arora, Avanika Narayan, Mayee F Chen, Laurel Orr, Neel Guha, Kush Bhatia, Ines Chami, Frederic Sala, and Christopher Ré. 2022. Ask me anything: A simple strategy for prompting language models. *arXiv preprint arXiv:2210.02441* (2022).

[2] David Baidoo-Anu and Leticia Owusu Ansah. 2023. Education in the era of generative artificial intelligence (AI): Understanding the potential benefits of ChatGPT in promoting teaching and learning. *Journal of AI* 7, 1 (2023), 52–62.

[3] Ali Borji. 2023. A categorical archive of chatgpt failures. *arXiv preprint arXiv:2302.03494* (2023).

[4] Julio C Caiza and Jose M Del Alamo. 2013. Programming assignments automatic grading: review of tools and implementations. *INTED2013 Proceedings* (2013), 5691–5700.

[5] Anita Carleton, Mark Klein, John Robert, Erin Harper, Robert Cunningham, Dionisio de Niz, John Foreman, John Goodenough, James Herbsleb, Ipek Ozkaya, Douglas Schmidt, and Forrest Shull. 2021. *Architecting the Future of Software Engineering: A National Agenda for Software Engineering Research Development.* https://insights.sei.cmu.edu/library/architecting-the-future-of-software-engineering-a-national-agenda-for-software-engineering-research-development/ Accessed: 2023-Dec-7.

[6] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. 2003. On automated grading of programming assignments in an academic institution. *Computers & Education* 41, 2 (2003), 121–131.

[7] Binglin Chen, Sushmita Azad, Rajarshi Haldar, Matthew West, and Craig Zilles. 2020. A validated scoring rubric for explain-in-plain-english questions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 563–569.

[8] Chase Geigle, ChengXiang Zhai, and Duncan C Ferguson. 2016. An exploration of automated grading of complex assignments. In *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*. 351–360.

[9] Wayne Holmes, Kaska Porayska-Pomsta, Ken Holstein, Emma Sutherland, Toby Baker, Simon Buckingham Shum, Olga C Santos, Mercedes T Rodrigo, Mutlu Cukurova, Ig Ibert Bittencourt, et al. 2021. Ethics of AI in education: Towards a community-wide framework. *International Journal of Artificial Intelligence in Education* (2021), 1–23.

[10] Stephen C Johnson. 1977. *Lint, a C program checker.* Bell Telephone Laboratories Murray Hill.

[11] Tiffany H Kung, Morgan Cheatham, Arielle Medenilla, Czarina Sillos, Lorie De Leon, Camille Elepaño, Maria Madriaga, Rimel Aggabao, Giezel Diaz-Candido, James Maningo, et al. 2023. Performance of ChatGPT on USMLE: Potential for AI-assisted medical education using large language models. *PLoS digital health* 2, 2 (2023), e0000198.

[12] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, et al. 2023. Augmented language models: a survey. *arXiv preprint arXiv:2302.07842* (2023).

[13] OpenAI. [n. d.]. *ChatGPT.* https://chat.openai.com/

[14] Michael Sheinman Orenstrakh, Oscar Karnalim, Carlos Anibal Suarez, and Michael Liut. 2023. Detecting llm-generated text in computing education: A comparative study for chatgpt cases. *arXiv preprint arXiv:2307.07411* (2023).

[15] James Perretta, Westley Weimer, and Andrew DeOrio. 2019. Human vs. automated coding style grading in computing education. In *2019 ASEE Annual Conference & Exposition*.

[16] Junaid Qadir. 2023. Engineering education in the era of ChatGPT: Promise and pitfalls of generative AI for education. In *2023 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 1–9.

[17] Sami Sarsa, Paul Denny, Arto Hellas, and Juho Leinonen. 2022. Automatic generation of programming exercises and code explanations using large language models. In *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*. 27–43.

[18] Johannes Schneider, Robin Richner, and Micha Riser. 2023. Towards trustworthy autograding of short, multi-lingual, multi-type answers. *International Journal of Artificial Intelligence in Education* 33, 1 (2023), 88–118.

[19] Johannes Schneider, Bernd Schenk, Christina Niklaus, and Michaelis Vlachos. 2023. Towards LLM-based Autograding for Short Textual Answers. *arXiv preprint arXiv:2309.11508* (2023).

[20] Kehui Tan, Tianqi Pang, and Chenyou Fan. 2023. Towards Applying Powerful Large AI Models in Classroom Teaching: Opportunities, Challenges and Prospects. *arXiv preprint arXiv:2305.03433* (2023).

[21] Eva AM Van Dis, Johan Bollen, Willem Zuidema, Robert van Rooij, and Claudi L Bockting. 2023. ChatGPT: five priorities for research. *Nature* 614, 7947 (2023), 224–226.

[22] Yaqing Wang, Quanming Yao, James T Kwok, and Lionel M Ni. 2020. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (csur)* 53, 3 (2020), 1–34.

[23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. arXiv:2201.11903 [cs.CL]

[24] Jules White, Quchen Fu, Sam Hays, Michael Sandborn, Carlos Olea, Henry Gilbert, Ashraf Elnashar, Jesse Spencer-Smith, and Douglas C Schmidt. 2023. A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382* (2023).

[25] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C. Schmidt. 2023. ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design. arXiv:2303.07839 [cs.SE]

[26] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL]

[27] Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910* (2022).

Received 7 December 2023; accepted 15 January 2024